1986

Performance Evaluation Modeling for Distributed Computing

Catherine E. Houstis

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
86-576

PERFORMANCE EVALUATION MODELING FOR DISTRIBUTED COMPUTING

Catherine E. Houstis
Elias N. Houstis
John R. Rice

CSD TR #86-576
January 1986

PERFORMANCE EVALUATION MODELING FOR DISTRIBUTED COMPUTING

Catherine E. Houstis *
Department of Electrical Engineering

Elias N. Houstis **
John R. Rice **
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

ABSTRACT

We present a methodology for evaluating the performance of application programs on distributed computing systems. An application A is represented by an annotated graph G(A) giving its requirements for processing, memory and communication plus the precedence between computation modules. Machines are represented by a similar graph G(M) and the methodology is to (a) map G(A) into G(M), (b) compress the application code to a much simpler one that uses the same resources, and (c) use a simulation package on the compressed code. Step (a) is subdivided into three parts (a1) reduction of the parallelism, (a2) scheduling the computational modules to (nearly) minimize communication costs, and (a3) actual layout of resulting graph into G(M). The two technical problems addressed here are (1) using communication delay models to simplify G(M) and the step (a3), (2) scheduling the modules (step (a2)). Our communication models apply well to "uniform" architectures, we explicitly consider the following five: single bus and common memory, single bus and distributed memory, multiple bus and distributed common memory, omega interconnection and distributed common memory, and omega interconnection and distributed memory.

* This research was supported by NSF grant DMC-8508684A1.

** This research was supported by ARO grant DAAG29-83-K-0026 and AFOSR grant 84-0385.

1. INTRODUCTION

In a previous paper [HOUS83] we considered the problem of mapping partial differential equation (PDE) computations into parallel machine architectures. We continue this study here, but with no particular involvement of PDE computations. We now consider the problem for a special, but widely used, class of architectures where we can obtain better results more simply. The architecture class we consider is where the machine has a somewhat homogeneous nature (this is made more specific later). To set the context, we briefly outline the general problem of mapping an application onto a machine and our approach to this problem.

We consider an application A to be a computation with four properties: processing requirements, memory requirements, communication requirements and precedence (or synchronization) between the subcomputations. Visualize the computation broken into *computational modules* which are nodes of a precedence graph for the computations. Note the processing and memory requirements at each node of the graph. Note the communication requirements along each link or edge of the graph. This annotated graph is called G(A). Note that links representing communication may need to be added to G(A) even though they are redundant as far as precedence is concerned.

We consider a machine to have three components: processing elements, memory elements and communication paths (an interconnection network). Similarly, the machine can be represented by an annotated graph G(M). In this paper, we consider the class of machines that can be modeled by a set of identical processors (which may have local memories), a set of identical common memories, and a queueing delay function which represents the communication performance characteristics of the machine. Five specific examples from this class are considered here. Intuitively, one may consider these as machines with a rather uniform architecture, one that scales simply with the number of processors and memories.

In general, the mapping problem has three somewhat independent steps:

1. Reduce the parallelism of the application to that of the machine.
2. Schedule the computational modules so that the application runs efficiently.
3. Given steps 1 and 2 are done, imbed the application into the machine.

The mapping problem is known to be NP complete [JENN77] and to find the optimal mapping for any of these three steps is also NP complete. We propose fast, heuristic algorithms here which are intended to produce good mappings at low cost. Due to the lack of space, we discuss the first step in depth elsewhere [HOUS86], a brief comment is given at the end about our approach. Given that steps 1 and 2 are done, step 3 is fairly easy for the class of machines considered here, see Section 4. Thus, this paper concentrates on the problem of scheduling the application so that it runs efficiently on one of the machines considered.

Our overall plan for making performance evaluations is as follows. We first devise fast, heuristic algorithms which carry out steps 1, 2 and 3 above. Note that the order

of steps 1 and 2 is not necessarily fixed, we assume in the discussions that step 1 precedes step 2 so as to simplify the discussion but this is not an essential ingredient to our approach. We use communication delay models for "uniform" architectures in order to simplify the complexity of the mapping problem. Once we have the application mapped onto the machine and scheduled to provide good efficiency, we then take the application code and "compress" it to a code that takes approximately the same resources. For example, the following loop

```
for i = 1 to 100
    for j = 1 to 20
       a = (i+j) * b
       x = 3. * cos(3.14 * a) + (a-b) * log(i+j+lmax)
    c(i,j) = x ** 2 + a/b
end
```

might be replaced by

```
for i = 1 to 2000
    dummy = 1. + 2. + 3. + 4. + 5. + 6. + 7. + 8. + 9. + 10. + 11. + 12.
    dummy = sin(a)+sin(b)
end
```

as there are 12 arithmetic and 2 function evaluations in the loop. Finally, we use a simulation package (for example, SIMON [FUJI85]) to "execute" the compressed code and estimate its performance.

The performance of a distributed system depends very much on how well the architecture and the algorithms are matched [KLEIN85]. A methodology for predicting multiprocessor performance from the systems point of view and the RP3 architecture is presented in [NORT85]. The problem of mapping has been studied in [BOKH81] and [BERM84,85] for the finite element machine and CHIP architectures respectively. In [WILL83] various objective criteria necessary for assigning processes to processors are examined and tested in a distributed environment. A different approach for modeling communications complexity of parallel algorithms is presented in [GANN86].

2. MODELING PERFORMANCE OF ALGORITHMS/ARCHITECTURE PAIRS

Assume that the graph G(A) is given and that an appropriate choice of problem size and granularity has been used in partitioning A to create G(A). Assume also that G(M) is given. In general, we want to map G(A) onto G(M) so the computational modules (nodes) of G(A) are associated with processors in G(M) and the communication links (edges) of G(A) are associated with data paths in G(M). We assume that the parallelism of G(A) is no larger than that of G(M) (see the discussion in Section 4). Normally the number of nodes in G(A) is much larger than the number of processors in G(M) and the computation is *scheduled* by assigning the computational modules to processors. This produces a new graph $G'(A)$. We choose $G'(A)$ to be efficient in the sense

that the communication costs of the computation are low. Experience shows that this criterion provides schedules which are also good in the sense of short elapsed execution times.

We assume that the machine is "uniform" in nature and richly endowed with communication paths. This assumption holds for many architectures including five specific cases considered here in detail. This assumption has two effects, first, it provides simple algorithms to carry out the final layout step of mapping G'(A) into G(M) (see the discussion in Section 4). Second, it allows us to model the communication behavior of the machine by known formulas that are practical to use in determining G'(A). In this section we introduce an example showing how G(A) appears and then present the communication delay models for the following architectures: (1) single bus and common memory, (2) single bus and distributed memory, (3) multiple bus and distributed common memory, (4) omega interconnection and distributed common memory, and (5) omega connection and distributed memory.

2.1 Application modeling

For the modeling of an application, we assume an initial partitioning of the computation into a number of communicating computational modules. If the communication paths among the modules are not known a priori (i.e real time applications) then the partition is modeled by a stochastic AND-EOR directed graph SG(A). This graph is specified by the following parameters:

m_i = processing time of module m_i
d_i = processing time of data block d_i
p_{ij} = probability of control transfer from m_i to m_j
a_{ij} = communication processing time between m_i and m_j

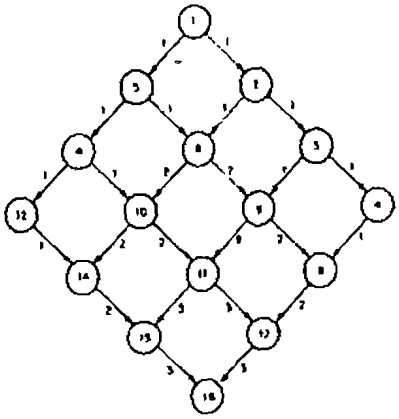
In [HOUS84] a stochastic analysis is applied to reduce SG(A) into a deterministic graph G(A) which comprises the input to the mapping model.

In case the communication paths among modules are known a priori then a dataflow language [FUJI85] is employed to specify the computation modules and their communication and synchronization requirements. We use the language SIMON [FUJI85] from the simulator for non-shared memory multiprocessor systems. Figure 1 presents the computation of each module in the example of a parallel Cholesky decomposition algorithm [O'LEA85]. Figure 2 depicts the corresponding graph G(A) and the values of the various workload parameters (node processing time and blocking time, communication traffic among nodes) obtained by setting SIMON's switching delay to zero. The *blocking time* or algorithm *synchronization delay* of a module is the time that

the module must wait for its inputs before its computation can start.

```
K:=0
WHILE (TRUE) BEGIN
   K: = K + 1 ;
   IF (K = i and K = j) then
      BEGIN
      X:=sqrt(X) ;
      IF (j ≠ N) PUT (OUT_EAST, X) ;
      IF (i ≠ N) PUT (OUT_SOUTH, X) ;
      BREAK;
      END
   ELSE IF (K=j) THEN
         BEGIN
         GET (IN-SOUTH, Y) ;
         X=X/Y ;
         IF (j ≠ N) PUT (OUT_EAST, X) ;
         IF (i ≠ N) PUT (OUT_SOUTH, Y) ;
         BREAK;
         END
   ELSE IF (K=j) THEN
         BEGIN
         GET (IN_EAST, Y) ;
         IF (j ≠ N) PUT (OUT_EAST, Y) ;
         IF (i ≠ N) PUT (OUT_SOUTH, X) ;
         BREAK;
   ELSE BEGIN
         GET (IN_SOUTH, Y) ;
         GET (IN_EAST, Z) ;
         X = X - Y * Z ;
         IF (i ≠ N) PUT (OUT_SOUTH, Y) ;
         IF (j ≠ N) PUT (OUT_EAST, Z) ;
         END
END WHILE
```

Figure 1. Computation of node (i,j) in SIMON dataflow language for a parallel Cholesky decomposition algorithm.



Module processing time
$\underline{m}$ = (24, 30, 30, 25, 27, 31, 37, 32, 27, 34, 38, 39, 21, 28, 35, 45)

Module blocking time
$\underline{b}$ = (0, 4, 14, 24, 15, 27, 37, 48, 33, 45, 61, 66, 51, 64, 76, 88)

Figure 2. Precedence graph G(A) of the parallel Cholesky decomposition algorithm for a 4x4 symmetric matrix. The assumed numbering of modules is specified in each node. The communication traffic is indicated as a weight on the links of the graph.

2.2 Communication Modeling in Multiprocessor Architectures

Our methodology for reducing the initial interconnection graph G(A) is based on minimizing its communication cost. Thus, it is crucial to be able to predict the communication *cost* of G(A) running on a specific multiprocessor system architecture. We consider architectures for which there is a realistic simple model of the communication cost at the message passing level. Such performance models of multiprocessor systems are given in [MAR83], [KRU83]. The performance measure is a delay function which provides the expected queueing delay a message suffers in traversing the interconnection network of the system. Queueing delay is computed as a function of the system's communication interconnection network utilization.

Let

$c_{i,j}$ = total traffic (# of messages) between modules m_i and m_j,

k,ℓ = processors assigned to computational modules, m_i, m_j

and $d_{k,\ell}$ = interprocessor distance in the interconnection network of the considered architecture,

then the *interprocessor communication cost* or queueing delay is given by

$$D = D(u)$$

where $u = u(c_{i,j}, d_{k,\ell}$, interconnection network characterization) denotes the utilization. When two modules are assigned to different processors their utilization of the interconnection network is computed as follows:

$$u = \frac{c_{i,j} * d_{k,\ell}}{C * T}$$

where C = capacity of interconnection network and T = time frame during which each parallel processor is running.

The delay obtained is only approximate because of the various simplifying assumptions involved in the analytical modeling of the performance characteristics (queueing, utilization) of the multiprocessor system. In all of the models we consider, two assumptions are made: (a) every processor accesses the interconnection network at the same rate, and (b) every processor accesses the various common memory modules with the same probability. These assumptions lead to a queueing model which is mathematically tractable. Intuitively, these assumptions imply that the communication requirements of G(A) are nearly uniform. If G(A) is one of many applications running in the same system, these assumptions may be realistic even if the algorithm has somewhat non-uniform communication patterns. A precise investigation on this question is under way.

We consider the performance analysis of five different multiprocessor parallel architectures; (1) Single bus, common memory, (2) Single bus, distributed common memory

modules, (3) Multiple bus, distributed common memory modules, (4) Omega interconnection, distributed memory modules, (5) Omega interconnection processor to processor multiprocessor system. The queueing delay functions of these systems depend on a number of parameters. Some of these parameters are (a) the number of processors in the system (b) the number of memory modules (c) the number of busses or switch size of the interconnection (d) the distribution of message size in the system (e) the message generation rate, (f) the memory access time, (g) the number of network stages.

Next, we present the queueing delay functions of these multiprocessor parallel architectures and the way they have been applied to the mapping algorithm.

(1) Single bus common memory architecture

In this architecture, (Figure 3(i)), k processors (P_i) are connected to an external common memory (CM) via a global bus (GB). Each P_i has a local memory (LM_i) connected to its own local bus (LB_i). The system has been analyzed in [MAR83] as a "machine repairman" problem. For given bus utilization level, say u_1, the average queueing delay per information transfer unit is given by $D_1(u_1) = (k-u_1/\rho_1)/u_1$. The parameter ρ_1 characterizes the communication of the architecture and it is found by solving the nonlinear algebraic equation

$$u_1 = \frac{P_k(\rho_1)-1}{P_k(\rho_1)} \quad \text{where } P_k(\rho_1) = \sum_{j=0}^{k} \rho_1^j k!/(k-j)!.$$

The workload characterization of the application is given by $\rho_p = \rho_1/2$ and represents the ratio λ_p/μ where λ_p is each P_i's access rate to the bus which in turn depends on the communication pattern of G(A) and $1/\mu$ is the average memory transfer requirement which depends on the average message length.

(2) Single bus distributed memory architecture

This architecture, (Figure 3(ii)), is obtained from the previous one by distributing the common memory to each processor. The local memory of each P_i is logically divided in private memory PM_i and common memory CM_i and accessed by a double port.

This system has been analyzed in [MAR83]. The average queueing delay per information transfer unit as a function of the bus utilization is given by

$$D_2(u_2) = \frac{k-u_2/\rho_2}{u_2} \quad \text{where } \rho_2 = \frac{\rho_p}{\rho_{p+1}}$$

$$u_2 = \frac{P_k(p_2)-1}{P_k(p_2)} \quad \text{where } P_k(\rho_2) = \sum_{j=0}^{k} \rho_2^j \frac{k!}{(k-j)!}$$

(3) Multiple bus distributed common memory modules

A multiple bus distributed common memory modules is shown in Figure 3(iii). P_i's have their own LM_i's and communicate via common memory modules. Multiple global busses are used by the P_i's to access CM_i's. Such a system is referred to as a k×m×b system where k is the number of P_i's, m is the number of CM_i's and b is the number of global busses used. We assume that $k \geq m > b$.

This system has been analyzed in [MAR82] using several approximate models based on a Markov chain approach. The model we are adopting here, provides performance characteristics which are an upper bound to the characteristics of the actual system.

The average queueing delay per information transfer unit is given by

$$D_3(u_3) = \frac{k - u_3/\rho_3}{u_3} \qquad \text{where } \rho_3 = \rho_p$$

$$u_3 = \frac{P_k(\rho_3) - 1}{P_k(\rho_3)} \quad \text{where } P_k(\rho_3) = \sum_{i=1}^{k} \frac{\rho_3^{k-j} \frac{k!}{(i-1)!} \prod_{l=1}^{k-i} \beta_l^{-1}}{1 + \sum_{j=0}^{k-1} (\rho_3^{k-j} \frac{k!}{j!} \prod_{l=1}^{k-j} \beta_l^{-1})}$$

$$\text{and} \quad \beta_\ell = \frac{\sum_{j=1}^{b-1} j p_j(\ell) + b \sum_{j=0}^{\ell-b} [p_b(j+b) p_{m-b}(\ell-2b-j+m)]}{\sum_{j=1}^{b-1} p_j(\ell) + \sum_{j=0}^{\ell-b} [p_b(j+b) p_{m-b}(\ell-2b-j+m)]}, \quad \ell \geq 0$$

where

$$p_j(\ell) = p_j(\ell-j) + p_{j-1}(\ell-j) + \cdots + p_1(1-j) + p_0(1-j)$$

with initial conditions

$$\begin{aligned} p_j(\ell) &= 0 \quad \ell < j \\ p_0(\ell) &= 0 \quad \ell > 0 \\ p_j(\ell) &= 1 \quad j \geq 0 \end{aligned}$$

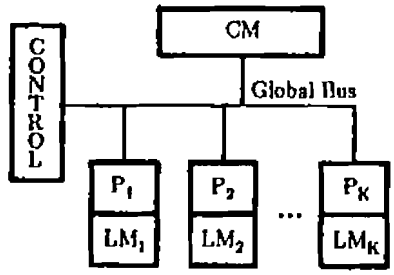
(4) Omega interconnection distributed memory modules

In Figure 3(iv), an example of an Omega interconnected architecture is shown where the Omega network is drawn for $k = 8$ processors in the system, and 2x2 switches.

This system has been analyzed by [KRU83], as the interconnection network of the ultracomputer [KRU83]. The same analysis applies to a general class of multistage interconnection networks called Banyan networks [KRU83]. Some of these are the Omega, Delta or indirect cube interconnection networks. In our case, we have

Architecture **Delay function**

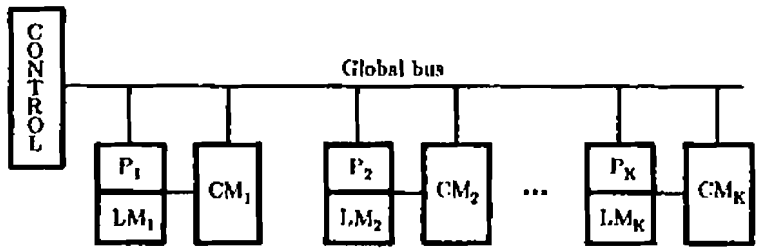
(i) Single bus common memory



$\rho_p = \rho/2$

$D = (k-u/\rho)/u$

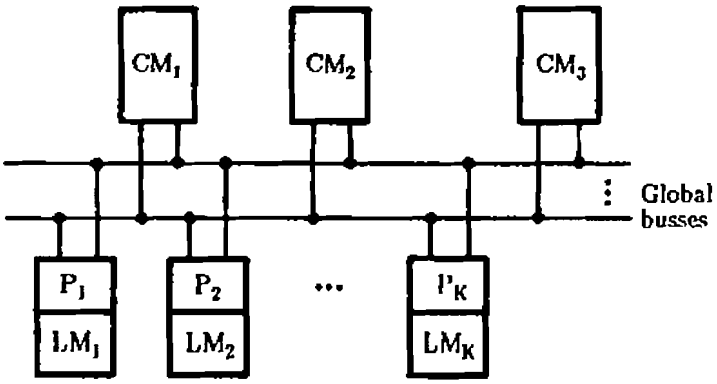
(ii) Single bus distributed common memory



$\rho_p = \frac{\rho}{1-\rho}$

$D = (k-u/\rho)/u$

(iii) Multiple bus distributed common memory



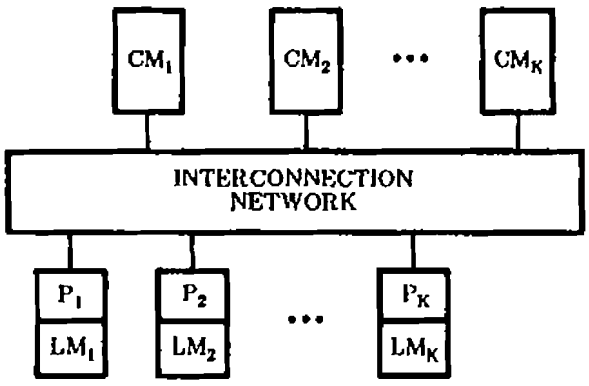
$\rho_p = \rho$

$D = (k-u/\rho)/u$

(iv) Omega Interconnection distributed common memory

$\rho_p = \rho$

$D = \log_n k(t_r + t_c \frac{m^2(1-1/n)\rho}{2(1-n\rho)}) + (m-1)t_r$



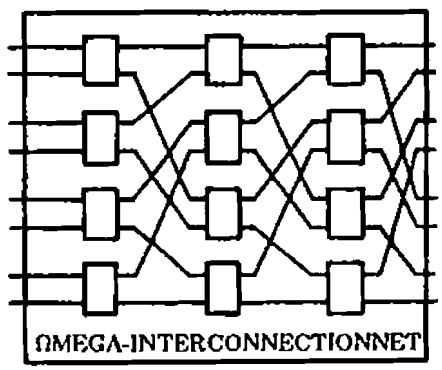


Figure 3. The delay function and workload parameter (ρ_p) as a function of utilization (u) and parameter (ρ) for four shared memory architectures with k processors. In case (iii) m = # of common memories, b = # of common busses and $k \geq m > b$. In case (iv) nxn is the size of the switch, t_r = transit time of a switch, t_c = cycle time of a switch, m = # of packets in a msg; ρ = average # of msgs entered by each P_i/cycle time.

considered buffered nxn switches where the capacity of the buffers is infinite. Then the average delay per message in traversing the interconnection network is simply its delay per stage times the number of stages ($\log_n k$) in the interconnection.

$$D_4(\rho_4) = \log_n k(t_r + t_c \frac{m^2(1 - \frac{1}{n})\rho_4}{2(1 - m\rho_4)}) + (m-1)t_c$$

and $$u_4 = \rho_4 x k$$

where k = number of processors in the system
t_r = transit delay of a switch
t_c = cycle time of a switch
n = size of switch (nxn)
m = number of packets per message
ρ_4 = average number of messages entered by each P_i per cycle time

An exact calculation of $D_4(\rho_4)$ is possible since $\rho_4 = u/k$ and can be directly substituted into the delay function.

(5) Omega Interconnection PE to PE Architecture

In every architecture considered so far $d_{k,\ell} = 1$. Figure 4 shows an architecture for which $d_{k,\ell} \neq 1$. The organization of the processors and memories differs from that in Figure 3(iv) in that each memory is associated with each processor forming a single element called the processing element (PE). Thus a PE to PE configuration is shown in Figure 4. Note that this time the distance between the processors is $d_{k,\ell} = 1,2$. For the average queueing delay per message through the omega interconnection a formula is given which is the same as in architecture (4) but it is stated in more general terms by [NOR85] and is as follows:

$$D = \frac{u}{2(1-u)} \times m \times st \times (1 - \frac{1}{u})$$

where u = utilization of the interconnection by a processor per cycle time
m = number of packets per message
st = number of network stages
nxn = size of switch

The above formula is applicable for the general class of Banyan interconnection networks, delta, omega, indirect cube.

Architecture **Delay function**

$$D = \frac{u}{2(1-u)} \times m \times st \times (1 - \frac{1}{u})$$

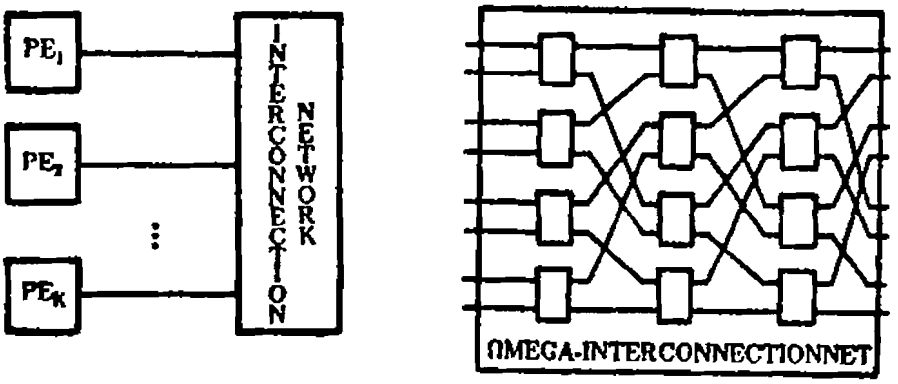


Figure 4. The delay function for nonshared memory interconnection networks u = utilization of the interconnection by a processor per cycle time, m = number of packets per message, st = number of network stages, nxn = size of switch.

3. CLUSTERING TO OPTIMIZE COMMUNICATION COSTS

We now consider how to obtain the schedule graph G'(A) from the original graph G(A). We use a heuristic algorithm based on the minimization of communication cost. This cost is estimated from the data about communication in G(A) and the average communication delay models such as given in Section 2. The details of this algorithm are given in [HOUS83] and we briefly summarize it here for completeness. There are numerous constraints implicit in the mapping problem which complicate a complete mathematical formulation considerably. However, experience shows that this algorithm runs fast and the run time is normally linear in the size of G(A) and the number of processors. The reduction can be viewed as a clustering process that tries to minimize *communication time for data and computation variables* by clustering modules together. This clustering must satisfy the following constraints:

(i) *resource constraints:*

- Every computational module must fit into the memory assigned
- Data blocks must fit into the memory assigned.
- Computations must have enough processor time.

(ii) *parallelism constraint:*

- Parallel computational modules can not be clustered, they are assigned to different processors

(iii) *artificial constraint:*

- Processing time on each processor for application A is limited to T (a parameter).

It is worth noting that the *time frame* parameter T is used implicitly to calculate the utilization (u) of the interconnection network of the machine M due to intermodule communication traffic of the application A. The reduction of T forces more and more processors to be used.

3.1 Clustering algorithm

The solution of the reduction problem for a specific time frame T is achieved by the following *heuristic clustering strategy:*

Start

Assign one processor per computation module

Assign data blocks to 'closest' memory

Interation

1. Select a pair of computation modules for possible merger into one processor which gives maximum reduction in objective function (communication cost).
2. If no constraints are violated, then merges the two modules, otherwise remove the pair from list of eligible pairs.

Experience suggests that this heuristic strategy "solves" the reduction problem in approximately linear time. The input to the clustering algorithm is as follows:

Algorithm specification

- the application graph G(A). Recall that this graph explicitly contains requirements for processing, memory and communication plus the precedence of the computation modules. From this information one may easily derive the synchronization delay or blocking times of the modules.

Architecture characteristics

- communication models (delay function)
- interconnection network bandwidth
- G(M) if different than G(A)

Resource constraint

- time frame parameter T

The output of the clustering algorithm for the Cholesky decomposition graph of Figure 2 are shown in Figures 5-7. Figure 5 is for a 4 by 4 matrix and T = 464 while Figures 6 and 7 are for a 5 by 5 matrix and use time frame parameters T = 541 and 473, respectively.

Processor number	Processor utilization	Modules/data blocks allocated
1	78.80%	1,2,3,4,11,12
2	38.20%	5,9,13
3	47.07%	6,7,8
4	89.94%	10,14,15,16

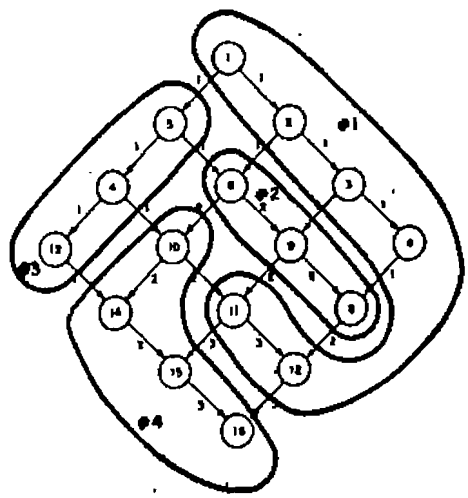


Figure 5. Mapping of computation modules for Cholesky decomposition algorithm to architecture 2 with 17 processors for time frame, T = 464. The utilization of each processor is given.

Processor number	utilization
1	76.26%
2	78.33%
3	46.83%
4	89.07%
5	51.84%

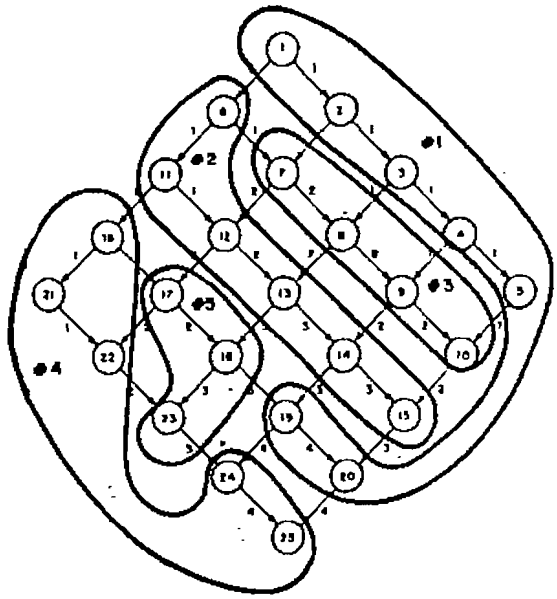


Figure 6. The solution of clustering algorithm for T = 541 and Cholesky decomposition algorithm for a 5x5 matrix. The module processing plus blocking time requirements, in the order of nodes specified, are 24, 34, 44, 54, 60, 42, 58, 74, 85, 90, 60, 79, 99, 195, 120, 78, 98, 117, 139, 145, 91, 190, 129, 151, 173.

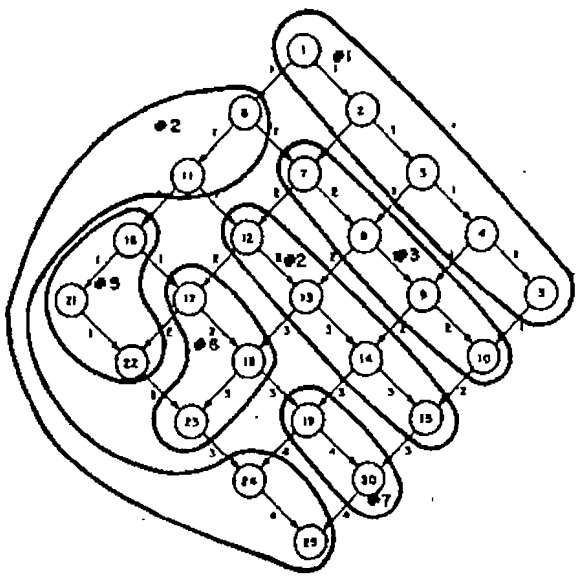


Figure 7. The solution of the clustering algorithm for T = 473 for the application considered in Figure 6.

3.2 On the Solution of the Graph Reduction Problem

When the time frame T changes, a different clustering in G(A) might be obtained as indicated in Figures 6 and 7. Throughout we denote by

T_{PAR}: the shortest time frame for which all allocated processors can run the application A in parallel,

T_{MIN}: shortest time frame for which the application A can run under the imposed resource utilization constraints.

Our observation has been that $T_{MIN} \neq T_{PAR}$ when the intermodular communication cost is very low. For example, in the case of Figure 2 there are ten different clusterings which are summarized in Figure 8. Normally, $T_{MIN} = T_{PAR}$ and the clustering algorithm produces a unique solution. If we define as an *optimum clustering* for G(A) as the one with minimum elasped time and minimum communication cost then it is easy to verify the following observations.

Lemma: If $T_{MIN} \neq T_{PAR}$ then the optimum clustering is the one that corresponds to time frame $T = T_{PAR}$ or the one with the minimum number of clusters.

For the different clusterings G'(A) of Figure 8, the elapsed time versus the time frame T are plotted in Figure 9. These results were obtained using the SIMON simulator.

The current implementation of the graph reduction phase is capable, by using an iterative procedure, to identify all possible solutions and the breakpoints as in Figure 8. Also, it is possible to estimate the elapsed time provided we are able to predict the blocking time of each modular in G(A) due to the precedence of computations. The workload analysis based on SIMON provides this information. It turns out that T_{PAR} is a close *upper bound* of the elapsed time when the blocking time of each module is incorporated as part of the processing time of the modules. We believe that this approach is reasonable since the blocking time is solely an attribute of the application algorithm. The importance of the elapsed time is twofold. (a) For the same application, the performance of different multiprocessor system architectures can be compared by comparing their elapsed time. The advantage of parallel processing can also be investigated by comparing the elapsed time of an application running on a parallel architecture system to a uniprocessor system. (b) Given different initial partitions G(A) of the same application A, their performance can be compared by looking at the elapsed time of the different partitions running on the same multiprocessor system.

4. REDUCTION OF PARALLELISM AND LAYOUT

In general, the degree of parallelism of a computation can be much larger than the number of processors available in G(M). We may either reduce the parallelism of G(A) initially or reduce the parallelism of G'(A) after the clustering. It is not a priori clear which is the best strategy, but similar techniques can be used in either case. For this purpose a number of heuristics have been devised. The analysis and performance of these algorithms are reported elsewhere [HOUS86]. We illustrate the nature of these

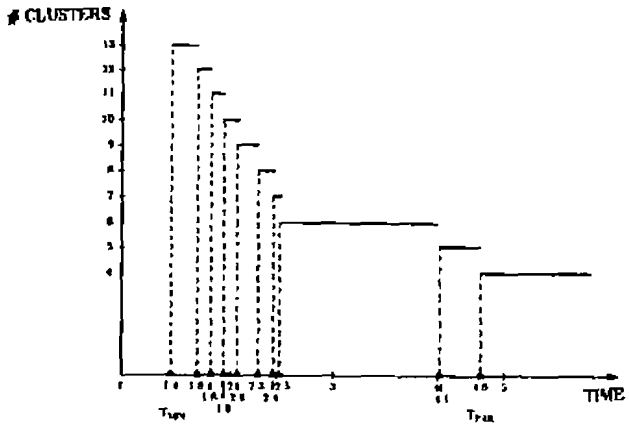


Figure 8. The number of clusters obtained for different values of time frame T for the Cholesky decomposition application, (4x4 matrix).

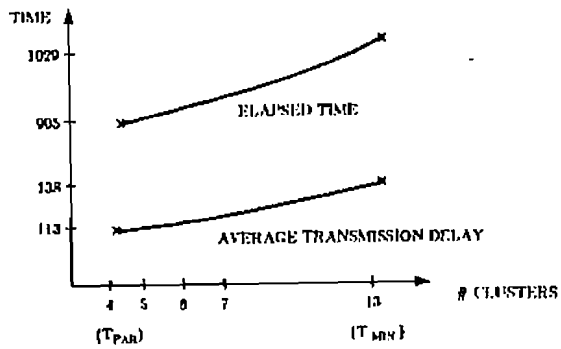


Figure 9. Elapsed time and average transmission delay of the different G'(A)'s obtained for the Cholesky decomposition application, (4x4 matrix).

algorithms by giving the simplest and the most promising.

Heuristic H_1: *Merge small pairs of parallel modules*

- start with all parallel modules G'(A) (output of clustering algorithm)
- Apply the clustering algorithm, after relaxing the parallelism constraint and ordering the pairs according to the sum of their processing times.
- Stop when the parallelism is reduced to that of G(M).

The motivation for this heuristic is that it makes use of an existing algorithm, it is fast and we hope it results in a small increase of the elapsed time.

Heuristic H_2: *Merge small pairs in sets with maximum parallelism.*

- Determine all sets of modules with maximum parallelism.
- Merge the pairs with the smallest sum of processing times.
- Repeat first two steps until maximum parallelism is acceptable.

In this case there is significant initial algorithmic cost (but still linear in the size of the graph). The cost per repetition is small and the parallelism is reduced rapidly. One can equally well merge pairs which decrease most the communication cost.

The *layout problem* is often taken to mean the whole process of mapping G(A) to G(M) or to substantial portions of it. In our approach we have obtained G'(A) which (a) has the same or less parallelism than G(M), and (b) has nearly minimal communication cost. There can still be substantial difficulties in mapping G'(A) into G(M) if the machine has a low level of interconnection (e.g., as in a ring or grid of processors). However, the class of architectures considered in this paper tend to have a high level of interconnection, so the final layout is not normally difficult. See [BERM84] for a more general discussion of the layout problem. We present simple layout algorithms for the five multiprocessor architectures described in Section 2.

1. *Single bus, common memory*
   Assign modules to processors in any manner consistent with G'(A).
2. *Single bus, distributed memory*
   Multiple bus, distributed memory
   Omega interconnection, distributed common memory
   Assign modules to processors so that data sets associated with particular modules are assigned to memories associated with the corresponding processors.
3. *Omega interconnection, distributed memory*
   Assign one module M_1 and corresponding data set to a processor P_1 and corresponding memory. Assign the module (and corresponding data set) that is nearest to M_1 to the processor that is the nearest neighbor to P_1 (and corresponding memory). Repeat until all modules are assigned.

4. References

[BERM84] Berman, F. Snyder, L., "On mapping parallel algorithms in parallel architectures," *Proceedings of the International Conference on Parallel Processing,* 1984, pp. 307-309.

[BERM85] Berman, F., Goodrich, M., Koelbel, C., Robinson, III, W. J., Showell, K., "Prep-p: A mapping preprocessor for chip computers," *Proceedings of the International Conference on Parallel Processing,* 1985, pp. 731-733.

[BOKH81] Bokhari, Shamid, H., "On the Mapping Problem," *IEEE Transactions on Computers,* Vol. C-30, No. 3, 1981, pp. 207-214.

[FUJI85] Fujimoto, R. M., "The SIMON simulation and development system," *Summer Computer Simulation Conference,* 1985 (Univ. of Utah).

[GANN86] Gannon, D. and J. Von Rosendale, On the communication complexity of parallel numerical algorithms, *IEEE Trans. on Computers,* to appear.

[HOUS86] Partitioning PDE algorithms: Methods and Performance Evaluations, *Parallel Computing,* 1986, to appear.

[HOUS84] Houstis, Catherine E., "Allocation of Real Time Application in Distributed Systems," submitted for publication to the *IEEE Transactions on Computer,* 1984.

[HOUS83] Houstis, C. E., Houstis, E. N., Rice, J., "Partitioning and Allocation of PDE Computation to Distributed Systems," *PDE Software: Modules Interfaces and Systems,* Edited by B. Enguist and T. Smedsaas, North Holland, 1983, pp. 67-85.

[HWAN84] Hwang, Kai, Briggs, Fayé, *Computer Architecture and Parallel Processing,* McGraw-Hill, 1984.

[JENN77] Jenny, C. J., "Process Partitioning on Distributed Systems," Digest of paper *National Telecommunications Conference,* 1977, pp. 31:1-31:10.

[KLEI85] Kleinrock, Leonard, "Distributed Systems," *Communications of ACM,* Vol. 28, Number 11, 1985, pp. 1200-1213.

[KRUS83] Kruskal, C., Snir, M., "The performance of multistage Interconnection Nets for multiprocessing," *IEEE Transactions on Computers,* Vol. C-32, No. 12, 1983, pp. 1091-1098.

[MARS83] Marsan, M. A., Gerla, M., "Markov models for Multiple Bus Multiprocessor Systems," *IEEE Transactions on Computers,* Vol. C-32, No. 3, 1983, pp. 239-248.

[MARS83] Marsan, M. A. Balbo, G., Conte, G., "Comparative Performance Analysis of Single Bus Multiprocessor Architectures," *IEEE Transactions on Computers,* Vol. C-31, No. 3, 1983, pp. 1179-1191.

[NORT85] Norton, Alan and G. F. Pfister, "A methodology for Predicting Multiprocessor Performance," *Proceedings of the International Conference on Parallel Processing,* 1985, pp. 772-778.

[O'LEA85] O'Leary, D. P. and G. W. Stewart, "Data-flow algorithms for parallel matrix computations," *Communication of ACM,* Vol. 28, 1985, pp. 840-853.

[WILL83] Williams, E. A., "Assigning Processes to Processors in Distributed Systems," *Proceedings of the International Conference on Parallel Processing,* 1983, pp. 404-406.